การเขียนโปรแกรมสำหรับการคำนวณแบบขนาน ตอน 3 การหาตัวเลขพรามแบบขนาน

จินต์ - นุดี วรมหาภูติ

ในฉบับที่ 39 นั้น ผู้เขียนได้แนะนำระบบคอมพิวเตอร์แบบหลายโปรเซสเซอร์แบบต่าง ๆ และบรรยายเกี่ยวกับเทคนิคการเขียนโปรแกรมเพื่อการคำนวณแบบขนาน สำหรับเครื่องระบบต่าง ๆ ที่แนะนำมา ท้ายสุด ผู้เขียนได้ยกตัวอย่างการเขียนโปรแกรมแบบง่าย ๆ สำหรับเครื่องคอมพิวเตอร์ดังกล่าว ในฉบับนี้ผู้เขียนจะยกตัวอย่างปัญหาที่เป็นจริงและเอาไปใช้ประโยชน์ได้ คือการคำนวณหาตัวเลขพราม (Prime number) ทุกตัวในช่วงที่กำหนดให้

ก่อนที่จะพูดถึงอัลกอริทึมสำหรับการหาตัวเลขพรามนี้ ผู้เขียนจะขอท้าวความเกี่ยวกับเทคนิคที่มีใช้กันอยู่ในปัจจุบันเสียก่อน เพราะผู้เขียนคิดว่าพวกเราส่วนใหญ่ยังไม่รู้วิธีหาตัวเลขพรามที่มีประสิทธิภาพทั้ง ๆ ที่เราเรียนรู้เกี่ยวกับตัวเลขพรามมาตั้งแต่เราเป็นเด็ก เรารู้เพียงแต่ว่าตัวเลขพรามคือตัวเลขจำนวนเต็มที่ไม่สามารถแยกตัวประกอบได้ ผู้เขียนยังจำได้ดีว่า โปรแกรมแรกที่เขียนคือ โปรแกรมหาตัวเลขพรามทุกตัวในช่วง 1 ถึง 1,000 เนื่องจากไม่รู้จักวิธีหาเลขพรามที่มีประสิทธิภาพมาก่อน โปรแกรมที่เขียนจึงใช้หลักการง่าย ๆ คือ ตัวเลขใดในช่วง 1 ถึง 1,000 จะเป็นเลขพรามก็ต่อเมื่อไม่สามารถหารลงตัวด้วยเลขจำนวนเต็มบวกทุกตัวที่น้อยกว่าตัวเลขนั้น (ยกเว้น 0 และ 1) จากหลักการนี้ เราสามารถเขียนโปรแกรมได้โดยใช้ loop 2 อันซ้อนกัน โดยที่ loop นอกจะเป็นตัวเลขที่ต้องการพิสูจน์ว่าเป็นเลขพรามหรือไม่ ส่วน loop ในจะเป็นช่วงการพิสูจน์โดยเพิ่มค่าตัวหารขึ้นจาก 2 ทีละหนึ่งจนกว่าจะเจอตัวที่หารลงตัวหรือว่าค่าของตัวหารเพิ่มขึ้นจนเท่ากับตัวถูกหารถ้าตัวเลขนั้นเป็นพราม

รูปที่ 14 เป็นตัวอย่างโปรแกรมที่ใช้หลักการที่อธิบายข้างต้นในการนับจำนวนตัวเลขพรามทั้งหมดในช่วง 2 ถึง 10 ล้าน จากการคำนวณแบบคร่าว ๆ ถ้าเราใช้เครื่องคอมพิวเตอร์ที่สามารถหารได้เร็วหนึ่งแสนครั้งต่อวินาที (ไมโครคอมพิวเตอร์โดยทั่วไปหารได้ช้ากว่านี้) เวลาที่ใช้คำนวณโปรแกรมในรูปที่ 14 จะประมาณ 1 ปีครึ่ง

หมายเหตุ นักคณิตศาสตร์บางคนนับเลขหนึ่งเป็นพราม แต่บางคนไม่นับ แล้วแต่คำนิยามของตัวเลขพรามที่แต่ละคนใช้ อย่างไรก็ตามทั้งสองนิยามไม่แตกต่างกันนัก สำหรับโปรแกรมทั้งหมดในบทความตอนนี้จะไม่ถือว่าเลขหนึ่งเป็นเลขพราม

จริง ๆ แล้ว วิธีการหาตัวเลขพรามทั้งหมดในช่วงที่กำหนดให้ถูกค้นพบมานานกว่าสองพันปีมาแล้ว โดย อีราโทส-ทีเนส (Eratosthenes) นักปราชญ์สมัยกรีก วิธีนี้รู้จักกันในนามของ **ซีฟของอีราโทสทีเนส (Sieve of Eratosthenes)** นับตั้งแต่ยุคนั้นมาจนถึงปัจจุบัน นักคณิต-ศาสตร์ได้สนใจค้นคว้าและวิจัยความลึกลับของตัวเลขพราม เพราะตัวเลขพรามเปรียบเสมือนหัวใจของคณิตศาสตร์ ยิ่งค้นคว้าตัวเลขพรามได้มากเท่าไร เราก็สามารถเรียนรู้ปัญหาต่าง ๆ ทางคณิต-ศาสตร์ได้มากขึ้นเท่านั้น อย่างไรก็ดี ซีฟยังคงเป็นที่ยอมรับในปัจจุบันว่าเป็นวิธีที่ดีที่สุดในการหาตัวเลขพรามทั้งหมดในช่วงที่กำหนดให้ เพียงแต่ได้รับแก้ไขการทำงานเพียงเล็กน้อยเพื่อให้มีประสิทธิภาพเพิ่มขึ้น น่าแปลกที่พวกเราส่วนใหญ่ไม่รู้จักซีฟที่ว่านี้ นอกจากนี้ในทางคอมพิว-เตอร์ นักวิจัยได้ใช้ซีฟเป็นโปรแกรมมาตรฐานในการเปรียบเทียบความสามารถของเครื่องคอมพิวเตอร์มานานพอสมควร เพราะซีฟสามารถใช้ทดสอบความเร็วของการอ่าน/เขียน array ขนาดใหญ่ได้

ซีฟของอีราโทสทีเนส

สมมติว่าเราต้องการหาตัวเลขพรามทุกตัวในช่วง 2 ถึง N การทำงานของซีฟ

```
PROGRAM findPRIME;
const
     N   =  10000000;
var
     i,   j,   K          :  integer;
     count                 :  integer;
     prime,    done        :  boolean;
begin
     count       := 0;
     for i       := 2 to N do
     begin
          prime  := true;
          done   := false;
          j      := 2;
          while  (j < i) and (not done) do
          begin
               K  := i mod j;
               if K = 0
               then begin
                         done  := true;
                         prime := false;
                    end;
               j  :=  j + 1;
          end;
          if prime
          then count := count + 1;
     end;
     writeln('Number of prime between 1 and ',N:7,' is',count:6);
end.
```

รูปที่ 14 โปรแกรมสำหรับนับจำนวนตัวเลขพรามทั้งหมดในช่วง 2 ถึง 10 ล้าน โดยใช้วิธีการหาร

จะเป็นไปตามขั้นตอนดังนี้คือ

1. เขียนตัวเลขทุกตัวในช่วง 2 ถึง N เรียงตามลำดับ

2. เริ่มที่เลข 2 โดยกระโดดไปตามลำดับของเลขทีละ 2 ก้าว เราจะกากบาทเลขทุกตัวที่เป็นฟหุคูณของ 2 ทิ้งไป ในกรณีนี้คือเลข 4, 6, 8...

3. หลังจากนั้น เราจะกลับมาที่จุดเริ่มต้น มองหาตัวเลขที่ยังไม่ถูกกากบาทถัดจากเลข 2 ซึ่งคือเลข 3 เราทำเช่นเดียวกับเลข 2 คือ กระโดดไปตามลำดับของเลขทีละ 3 ก้าว กากบาทเลขทุกตัวที่เป็นพหุคูณของ 3 ทิ้งไป ในกรณีนี้คือ 6, 9, 12...

4. หลังจากนั้น เราจะกลับมาทำตามขั้นตอนข้อ 3 โดยหาเลขที่ยังไม่ถูกกากบาทตัวถัดไป และกากบาทเลขทุกตัวที่เป็นพหุคูณของเลขนั้นทิ้งไป ทำซ้ำเช่นนี้จนกว่าจะไม่มีเลขที่ไม่ถูกกากบาทเหลือให้ใช้เป็นจุดเริ่มต้นอีก

5. หลังจากที่กากบาทตัวเลขทิ้งตัวเลขทุกตัวที่ไม่ถูกกากบาทจะเป็นตัวเลขพราม

ในรูปที่ 15 เป็นตัวอย่างการหาตัวเลขพรามทุกตัวในช่วง 2 ถึง 30 ถ้าสังเกตดูให้ดี เราจะเห็นว่าหลังจากกากบาทรอบที่สามเสร็จ ตัวเลขทุกตัวที่ไม่ถูกกากบาทจะเป็นตัวเลขพรามอยู่แล้ว ดังนั้น การกากบาทในรอบที่ 4 ถึง 6 จึงไม่มีประโยชน์ เพราะจะไปกากบาทซ้ำกับตัวที่ถูกกากบาทมาก่อนหน้านี้ สาเหตุเพราะตัวเลขที่เล็กกว่า N จะไม่สามารถมีตัวประกอบทุกตัวใหญ่กว่า $\sqrt{N}$ ได้ เช่น N = 30 ตัวเลขทุกตัวที่เล็กกว่า 30 ไม่สามารถจะมีตัวประกอบทุกตัวที่ใหญ่กว่า 5 ได้ ดังนั้น การกากบาทตัวคูณของ 7 จึงไม่จำเป็นเพราะ 14 และ 28 ถูกกากบาทในรอบแรก (พหุคูณของ 2) 21 ถูกกากบาทในรอบที่สอง (พหุคูณของ 3)

นอกจากนี้ ถ้าสังเกตให้ดี จะพบว่าการกากบาทพหุคูณของตัวเลขตัวใดสามารถเริ่มต้นที่ค่ายกกำลังสองของเลขนั้นได้ทันที ยกตัวอย่างเช่น การกากบาทพหุคูณของเลข 5 เราสามารถเริ่มต้นกากบาทได้ที่ 25 แทนที่จะเริ่มต้นทำที่ 10 เป็นต้นไป ทั้งนี้เพราะพหุคูณทุกตัวของ 5 ที่น้อยกว่า 25 ถูกกากบาททิ้งมาก่อนหน้านี้แล้ว เช่น 10 (5×2) และ 20 (5×4) ถูกกากบาททิ้งในรอบแรก เพราะเป็นพหุคูณของสอง ส่วน 15 (5×3) ถูกกากบาทในรอบที่สองเพราะเป็นพหุคูณของสาม

รูปที่ 16 เป็นตัวอย่างโปรแกรมที่ใช้หลักของซีฟในการนับจำนวนตัวเลขพรามทั้งหมดในช่วง 2 ถึง 10 ล้าน ซึ่งในที่นี้เราสมมติว่า เราสามารถสร้าง array ขนาด 10 ล้านตัวอักษรได้ ด้วยวิธีใหม่นี้ เราจะใช้เวลาคำนวณโปรแกรมในรูปที่ 16 เพียงไม่กี่นาที และไม่เกินหนึ่งชั่วโมงสำหรับเครื่องที่มีความสามารถต่ำ เราจะเห็นว่าประสิทธิภาพของซีฟดีกว่าวิธีการที่เราคุ้นเคยมากนัก ซึ่งแทบจะไม่น่าเชื่อว่าหลักการพื้นฐานง่าย ๆ เช่นนี้จะสามารถทำงานได้เร็วกว่าวิธีที่เรารู้จักเป็นพันเท่า

ซีฟแบบขนาน

หลังจากที่ผู้เขียนได้แนะนำโปรแกรมเพื่อหาตัวเลขพรามทุกตัวในช่วงที่กำหนดให้โดยใช้หลักการของซีฟไปแล้ว คราวนี้เราจะมาพูดถึงการเปลี่ยนโปรแกรมที่คำนวณแบบอนุกรมมาเป็นโปรแกรมที่คำนวณแบบขนาน โดยจะแสดงตัวอย่างสองวิธีของซีฟแบบขนาน พร้อมทั้งชี้

หมายเหตุ โปรแกรมที่แสดงในรูปที่ 16 เป็นเพียงตัวอย่างโปรแกรมที่ใช้หลักการของซีฟที่อธิบายข้างต้น ถ้าพิจารณาในแง่ประสิทธิภาพของโปรแกรม เรายังแก้ไขได้อีกหลายจุด อย่างน้อย เราสามารถแก้ไขโปรแกรมในรูป 16 ให้ทำงานได้เร็วขึ้น พร้อมทั้งประหยัดหน่วยความจำได้ประมาณอย่างละ 50 เปอร์เซ็นต์ แต่ผู้เขียนจะไม่กล่าวถึงวิธีการดังกล่าว เพราะเป็นจุดย่อยเกินไป จะขอทิ้งไว้เป็นปัญหาให้ขบคิดเล่น ๆ สำหรับผู้ที่สนใจ แต่จะบอกไว้ให้นิดว่าการแก้ไขดังกล่าวใช้หลักความจริงที่ว่า เลขคู่ทุกตัวยกเว้น 2 ไม่ใช่ตัวเลขพราม

การกากบาท

รอบที่ 1 (2)	รอบที่ 2 (3)	รอบที่ 3 (5)	รอบที่ 4 (7)	รอบที่ 5 (11)	รอบที่ 6 (13)	เลขพราม
2	2	2	2	2	2	2
3	3	3	3	3	3	3
✕	⊗	⊗	⊗	⊗	⊗	
5	5	5	5	5	5	5
✕	✕	⊗	⊗	⊗	⊗	
7	7	7	7	7	7	7
✕	⊗	⊗	⊗	⊗	⊗	
9	9	⊗	⊗	⊗	⊗	
✕	⊗	✕	⊗	⊗	⊗	
11	11	11	11	11	11	11
✕	✕	⊗	⊗	⊗	⊗	
13	13	13	13	13	13	13
✕	⊗	⊗	✕	⊗	⊗	
15	✕	15	⊗	⊗	⊗	
✕	⊗	⊗	⊗	⊗	⊗	
17	17	17	17	17	17	17
✕	✕	⊗	⊗	⊗	⊗	
19	19	19	19	19	19	19
✕	⊗	✕	⊗	⊗	⊗	
21	⊗	⊗	✕	⊗	⊗	
✕	⊗	⊗	⊗	✕	⊗	
23	23	23	23	23	23	23
✕	✕	⊗	⊗	⊗	⊗	
25	25	✕	⊗	⊗	⊗	
✕	⊗	⊗	⊗	⊗	✕	
27	✕	⊗	⊗	⊗	⊗	
✕	⊗	⊗	✕	⊗	⊗	
29	29	29	29	29	29	29
✕	✕	✕	⊗	⊗	⊗	

รูปที่ 15 การหาตัวเลขพรามทุกตัวในช่วง 2 ถึง 30 โดยใช้หลักการของซีฟ

× แสดงถึงตัวเลขที่ถูกกากบาททิ้งในรอบนั้น

⊗ แสดงถึงตัวเลขที่ถูกกากบาททิ้งมาก่อนหน้านี้

ข้อดีข้อเสียของวิธีดังกล่าว วิธีแรกที่จะกล่าวถึงถูกแนะนำโดย บอคคาริ (Bokhari) ส่วนวิธีหลังเป็นอัลกอริทึมที่ผู้เขียนคิดขึ้นมาเอง

วิธีที่บอคคาริเสนอแนะสำหรับซีฟแบบขนาน ทำงานดังต่อไปนี้คือ อัลกอริทึมสำหรับเครื่องที่มี P โปรเซสเซอร์ จะประกอบด้วย หนึ่งโปรเซสเจ้านาย (Master process) และ P โปรเซสทาส (Slave process) โดยที่โปรเซสเจ้านายจะคำนวณร่วมกับโปรเซสทาสตัวใดตัวหนึ่งบนหนึ่งโปรเซสเซอร์ ที่เป็นเช่นนี้เพราะบอคคาริมีเหตุผลบางประการที่ผู้เขียนจะกล่าวถึงต่อไป การทำงานของอัลกอริทึมเริ่มต้นจากโปรเซสเจ้านายแตกตัวออกเป็น P + 1 โปรเซส (หนึ่งโปรเซสเจ้านายและ P โปรเซสทาส) คำนวณแบบขนาน โปรเซสทาสทั้งหมดจะเริ่มต้นด้วยการเขียนค่าเริ่มต้น (Initialization) ของ array A ให้ทุกเซลมีค่าเป็นช่องว่าง (blank) การทำงานช่วงนี้จะเป็นแบบขนาน คือ ทุกโปรเซสทาสจะแบ่งงานทำกันคนละส่วน เพื่อให้การเขียนค่าเริ่มต้นเป็นไปได้เร็วขึ้น โดยแต่ละโปรเซสทาสจะเขียนค่าของ array A ประมาณ N/P เซลเป็นช่องว่าง ในระหว่างนั้นโปรเซสเจ้านายจะหยุดรอให้โปรเซสทาสเขียนค่าเริ่มต้นให้เสร็จสิ้นเสียก่อน การรอที่จุดนี้อาจจะใช้แนวกั้น (Barrier) ดังที่กล่าวมาในตอนที่แล้ว

เมื่อการเขียนค่าเริ่มต้นเสร็จสิ้นลง โปรเซสเจ้านายจะเริ่มทำงานโดยเคลื่อนตัวไปตาม array A เมื่อพบว่าเซลช่องใดของ array A เป็นช่องว่างอยู่โปรเซสเจ้านายจะส่งหมายเลขของช่องว่างนั้น (index) ไปยังโปรเซสทาสตัวใดตัวหนึ่งที่หยุดรออยู่ เมื่อโปรเซสทาสได้รับตัวเลขมาก็จะเริ่มกากบาทพหุคูณทุกตัวของเลขที่ได้รับมา โดยการเขียนค่า 'X' ลง

```
PROGRAM  serialSIEVE;
const
   N      =  10000000;
var
   buff                 : array [1..N] of char;
   i,j                  : integer;
   pointer, rootN       : integer;
   done                 : boolean;
begin
   rootN     := trunc(sqrt(N));
   for i     := 1 to N  do
      buff[i]   :=  ' ';
   buff[1]   := 'x';
   pointer   := 1;
   done      := false;
   while  (not done) do
   begin
        pointer := pointer + 1;
        while (buff[pointer] = 'x') and
              (pointer <= rootN)    do
               pointer  := pointer  + 1;
        if    (pointer > rootN)
        then   done := true
        else   begin
                  i  :=  pointer * pointer;
                 while (i <= N)  do
                 begin
                    buff[i]  := 'x';
                    i        := i + pointer;
                 end;
               end;
   end;
   j       := 0;
   for i   := 1 to N do
      if   buff[i] = ' '
      then j := j + 1;
   writeln ('Number of prime between 1 and',N:7,' is',j:6);
end.
```

รูปที่ 16 โปรแกรมสำหรับนับจำนวนตัวเลขพรามทั้งหมดในช่วง 2 ถึง 10 ล้าน โดยใช้หลักการของซีฟ

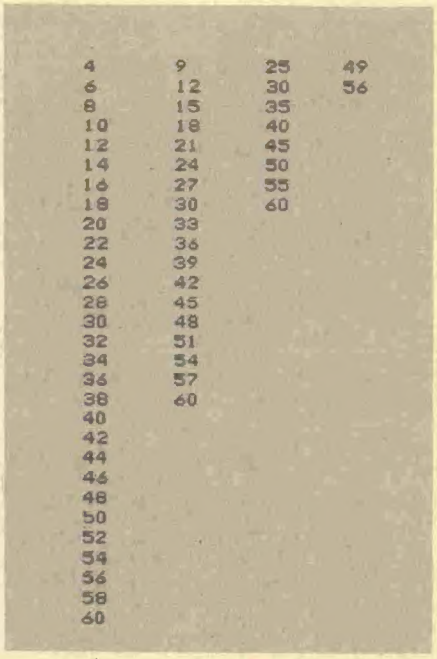
ในเซลทุกตัวที่มีหมายเลขประจำเซล (index) เป็นพหุคูณของเลขที่ได้รับมานั้น

สำหรับโปรเซสเจ้านาย เมื่อส่งตัวเลขตัวหนึ่งไปให้โปรเซสทาสแล้ว ตัวเองจะเคลื่อนตัวไปตาม array A จนกว่าจะพบเซลที่ยังเป็นช่องว่างอยู่ เมื่อพบก็จะส่งหมายเลขของเซลนั้นไปให้โปรเซสทาสที่หยุดรออยู่ ในกรณีที่โปรเซสทาสทุกตัวไม่ว่าง โปรเซสเจ้านายจะต้องหยุดรอจนกว่าจะมีโปรเซสทาสส่งสัญญาณมาบอกว่าทำงานหมดแล้ว โปรเซสเจ้านายจะได้ส่งงานใหม่ไปให้ทำ การทำงานจะวนไปจนกว่าโปรเซสเจ้านายจะเคลื่อนตัวผ่านเซลที่มีหมายเลขประจำตัวเป็น $\sqrt{N}$ ไปแล้ว

ที่นี้เรามาพูดถึงเหตุผลที่ให้โปรเซสเจ้านายคำนวณบนโปรเซสเซอร์ตัวเดียวกับโปรเซสทาสตัวใดตัวหนึ่ง เพราะหากโปรเซสเจ้านายเคลื่อนตัวไปตาม array A เร็วเกินไป จะเจอช่องว่าง ซึ่งที่จริงควรจะถูกกากบาท โดยทาสตัวใดตัวหนึ่งในช่วงเวลาอันใกล้ ดังนั้น โปรเซสเจ้านายจะเอาตัวเลขของเซลนั้นส่งไปให้โปรเซส-ทาสตัวอื่น เหตุการณ์ที่ว่านี้ไม่ทำให้ผลลัพธ์ที่ได้ออกมาผิดพลาด แต่จะทำให้เสียเวลาโดยเปล่าประโยชน์ ยกตัวอย่างเช่น โปรเซสเจ้านายเคลื่อนตัวผ่าน 2 และ 3 มาแล้ว ดังนั้น จะมีสองโปรเซส-ทาสกำลังกากบาทพหุคูณของ 2 และ 3 อยู่ แต่ถ้าโปรเซสเจ้านายเคลื่อนมาเร็วกว่าที่โปรเซสทาสจะเริ่มกากบาทเลข 4 ทิ้งไป โปรเซสเจ้านายจะพบว่าเซลช่องที่ 4 ยังคงว่างอยู่ ดังนั้น จะเอาเลข 4 ส่งให้โปรเซสทาสตัวอื่นไปกากบาทพหุคูณของ 4 ซึ่งที่จริงแล้วก็คือพหุคูณของ 2 เรียกว่าโปรเซสทาสตัวนั้นทำงานฟรี หากเราถ่วงเวลาให้โปรเซสเจ้านายเคลื่อนที่ช้าลง จะลดอัตราการทำงานซ้ำซ้อนกันให้น้อยลง เป็นผลให้ประสิทธิภาพส่วนรวมเพิ่มขึ้นได้ จากการทำงานที่อธิบายไปสามารถนำมาเขียนเป็นอัลกอริทึมได้ดังต่อไปนี้

โปรแกรมย่อยเจ้านาย (Procedure MASTER)

1. รอที่แนวกั้น (Barrier) จนกว่าโปรเซสทาสทุกตัวจะเขียนค่าเริ่มต้นเสร็จ
2. เคลื่อนไปตาม array A จนกว่าจะเจอเซลที่เป็นช่องว่าง ถ้าหมายเลขของเซลมากกว่า $\sqrt{N}$ ข้ามไปทำข้อ 6 ไม่เช่นนั้นทำงานต่อไป
3. ตรวจสอบดูว่ามีโปรเซสทาสว่างอยู่หรือไม่ ถ้าไม่มีให้รอจนกว่าจะได้รับสัญญาณมาจากโปรเซสทาสว่าทำงานเก่าเสร็จแล้ว
4. ส่งหมายเลขของเซลที่ได้จากข้อ 2 ไปให้โปรเซสทาสที่ว่าง
5. กลับไปทำข้อ 2
6. ส่งสัญญาณให้โปรเซสทาสทุกตัวเลิกทำงานได้ (ส่งเลขศูนย์ไปให้)
7. เลิกทำงาน



รูปที่ 17 การทำงานของโปรแกรมที่เขียนโดยใช้วิธีของบอคคาริในการหาตัวเลขพรามทุกตัวในช่วง 2 ถึง 60 ตัวเลขที่แสดงอยู่ในแนวตั้ง เป็นตัวเลขที่ถูกกากบาทโดยโปรเซสทาสตัวใดตัวหนึ่งในช่วงเวลาต่าง ๆ กัน

โปรแกรมย่อยทาส (Procedure SLAVE)

1. เขียนค่าเริ่มต้นของ array A ในส่วนของตัวเองให้มีค่าเป็นช่องว่าง
2. เข้าสู่แนวกั้นเพื่อส่งสัญญาณว่าเขียนค่าเริ่มต้นเสร็จ
3. รอรับหมายเลขที่จะกากบาท ถ้าเลขที่ได้รับเป็นศูนย์ข้ามไปทำข้อ 7
4. กากบาทพหุคูณทุกตัวของเลขที่ได้รับในข้อ 3
5. ส่งสัญญาณว่าทำงานเสร็จแล้ว
6. กลับไปทำข้อ 3
7. เลิกทำงาน

สำหรับตัวอย่างการหาตัวเลขพรามทุกตัวในช่วง 2 ถึง 60 โดยเครื่องสี่โปร-เซสเซอร์ได้แสดงไว้ในรูปที่ 17 การทำงานวิธีที่บอคคาริเสนอขึ้นมามีข้อเสียตรงที่ งานที่โปรเซสทาสแต่ละตัวได้รับจะไม่เท่ากัน เพราะจำนวนพหุคูณของเลขแต่ละตัวจะไม่เท่ากัน ถ้าเรามีจำนวนโปรเซสเซอร์มาก ๆ ปัญหาที่จะตามมาคือ โปรเซสทาสที่ได้รับเลข 2 ไปกากบาทจะทำงานมากที่สุด ในขณะที่โปรเซสทาสตัวอื่น ๆ ทำงานหมดไปนานแล้ว ทำให้เวลาในการคำนวณส่วนรวมเพิ่มขึ้นด้วย เนื่องจากในการวัดประสิทธิภาพของโปรแกรมสำหรับเครื่องหลายโปรเซสเซอร์ เราจะนับว่าโปรแกรมใช้เวลานานเท่าไรในการคำนวณ เริ่มต้นจากโปรแกรมเริ่มโหลด จนโปรแกรมทำงานทั้งหมดเสร็จ โดยไม่คำนึงถึงว่าระหว่างนั้นจะมีโปร-เซสเซอร์ว่างอยู่กี่ตัวก็ตาม จากตัวอย่างในรูปที่ 17 เราจะเห็นว่าวิธีของบอคคาริจะต้องใช้เวลาอย่างต่ำ 29 หน่วยในการกากบาทพหุคูณ นอกจากนี้วิธีที่บอคคาริเสนอขึ้นมายังมีข้อจำกัดตรงที่สามารถคำนวณได้เฉพาะบนเครื่องระบบหน่วยความจำร่วมกับระบบผสมเท่านั้น เนื่องจาก array A จำเป็นจะต้องอยู่ในหน่วยความจำส่วนกลาง เพื่อให้โปรเซสทาสทุกตัวสามารถใช้งานได้

เพื่อหลีกเลี่ยงปัญหาที่เกิดขึ้นในวิธี

ของบอคคาริ ผู้เขียนได้คิดอัลกอริทึมใหม่ขึ้นมา ในอัลกอริทึมใหม่นี้ แทนที่เราจะให้โปรเซสทาสตัวใดตัวหนึ่งกากบาทพหุคูณทุกตัวของเลขที่ได้รับ เราจะให้โปรเซสทุกตัวช่วยกันกากบาทพหุคูณของเลขทุกตัวที่โปรเซสเจ้านายส่งมาให้ นอกจากนี้โปรเซสเจ้านายในอัลกอริทึมนี้ก็ยังขยันทำงานไม่เอาเปรียบทาสเหมือนวิธีก่อน โดยที่โปรเซสเจ้านายจะช่วยโปรเซสทาสทำงานเท่า ๆ กันด้วย

การทำงานของอัลกอริทึมจะเป็นไปดังนี้คือ ในการหาตัวเลขพรามทุกตัวในช่วง 2 ถึง N โดยเครื่องคอมพิวเตอร์ที่มี P โปรเซสเซอร์เราจะแบ่ง array A ออกเป็น P ส่วนเท่า ๆ กัน ส่วนแรกจะถูกดูแลโดยโปรเซสเจ้านาย ส่วนที่สอง, สาม, สี่, ถึง P จะถูกดูแลโดยโปรเซส–ทาสตัวที่หนึ่ง, สอง, สาม, ถึง P-1 ตามลำดับ การทำงานจะเริ่มต้นที่โปรเซสทุกตัว (ทั้งเจ้านายและทาส) เขียนค่าเริ่มต้นของ array A ทุกเซลในส่วนที่ตนดูแลให้มีค่าเป็นช่องว่าง เมื่อทำงานในช่วงนี้เสร็จทุกโปรเซสจะเข้าสู่แนวกั้น (Barrier) เพื่อรอให้ทุกโปรเซสทำงานเสร็จหมดเสียก่อน ต่อจากนั้นโปรเซสเจ้านายจะเริ่มเคลื่อนตัวไปตาม array A มองหาเซลที่ยังเป็นช่องว่าง เมื่อพบเซลดังกล่าว โปรเซสเจ้านายจะส่งหมายเลขประจำเซลนั้น (index) ไปให้โปรเซสทาสทุกตัว เมื่อทุกโปรเซสได้รับหมายเลขมาแล้ว ทุกโปรเซส (รวมทั้งเจ้านาย) จะเริ่มกากบาทพหุคูณของเลขดังกล่าวที่อยู่ในส่วนของ array A ที่ตัวเองดูแลอยู่ หลังจากกากบาทเสร็จแล้ว ก็จะกลับไปรอที่แนวกั้นจนกว่าโปรเซสทุกตัวจะทำงานเสร็จ และรอให้โปรเซสเจ้านายมองหาตัวเลขต่อไปส่งมาให้อีก ทำวนไปจนกว่าโปรเซสเจ้านายเคลื่อนผ่านเซลที่มีหมายเลขมากกว่า $\sqrt{N}$ เป็นอันเสร็จงาน จากการทำงานที่อธิบายไปสามารถนำมาเขียนเป็นอัลกอริทึมได้ดังต่อไปนี้

โปรแกรมย่อยเจ้านาย (Procedure MASTER)

1. เขียนค่าเริ่มต้นของ array A ในส่วนที่ตัวเองดูแลให้ทุกเซลมีค่าเป็นช่องว่าง

2. เข้าสู่แนวกั้น (Barrier) รอให้ทุกโปรเซสทำข้อ 1 เสร็จ

3. เคลื่อนไปตาม array A จนกว่าจะเจอเซลที่เป็นช่องว่าง ถ้าหมายเลขของเซลมากกว่า $\sqrt{N}$ ข้ามไปทำข้อ 8

4. ส่งหมายเลขที่ได้มาจากข้อ 3 ไปให้โปรเซสทาสทุกตัว

5. กากบาทพหุคูณของเลขในข้อ 3 ในส่วนของ array A ที่ตัวเองดูแลอยู่

6. เข้าสู่แนวกั้น รอจนกว่าโปรเซสทุกตัวจะทำงานเสร็จ

7. กลับไปทำข้อ 3

8. ส่งสัญญาณให้โปรเซสทาสทุกตัวเลิกทำงานได้ (ส่งเลขศูนย์ไปให้)

9. เลิกทำงาน

โปรแกรมย่อยทาส (Procedure SLAVE)

1. เขียนค่าเริ่มต้นของ array A ในส่วนที่ตัวเองดูแลให้ทุกเซลมีค่าเป็นช่องว่าง

2. เข้าสู่แนวกั้น รอให้ทุกโปรเซสทำข้อ 1 เสร็จ

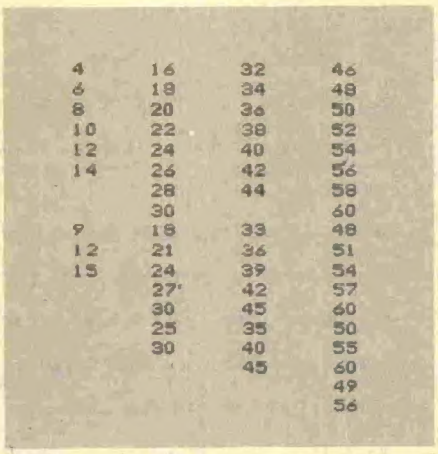
3. รอรับหมายเลขที่จะกากบาท ถ้าหมายเลขที่ได้รับเป็นศูนย์ข้ามไปทำข้อ 7

4. กากบาทพหุคูณของเลขที่ได้รับในข้อ 3 ในส่วนของ array A ที่ตัวเองดูแลอยู่

5. เข้าสู่แนวกั้นรอจนกว่าโปรเซสทุกตัวจะทำงานเสร็จ

6. กลับไปทำข้อ 3

7. เลิกทำงาน



รูปที่ 18 การทำงานของโปรแกรมที่เขียนโดยใช้วิธีของผู้เขียนในการหาตัวเลขพรามทุกตัวในช่วง 2 ถึง 60 ตัวเลขที่แสดงอยู่ในแนวตั้งเป็นตัวเลขที่ถูกกากบาทโดยโปรเซสตัวใดตัวหนึ่งในช่วงเวลาต่าง ๆ กัน

ตัวอย่างการหาตัวเลขพรามทุกตัวในช่วง 2 ถึง 60 โดยเครื่อง 4 โปรเซส–เซอร์ได้แสดงไว้ในรูปที่ 18 เราจะเห็นจากรูปที่ 18 ว่า อัลกอริทึมที่ผู้เขียนคิดขึ้นมาใช้เวลาในการกากบาทเพียง 18 หน่วย ซึ่งน้อยกว่าอัลกอริทึมที่บอคคาริเสนอมา อย่างไรก็ตาม วิธีที่ผู้เขียนเสนอมายังมีจุดอ่อนคือ ในความเป็นจริงค่าโสหุ้ยที่ใช้ปล่อยโปรเซสในแนวกั้น (Barries) หลังจากที่โปรเซสสุดท้ายเข้ามาถึงมีมากพอสมควร ทำให้ประสิทธิภาพโดยส่วนรวมของโปรแกรมลดลง อย่างไรก็ดี ผลการทดลองอัลกอริทึมทั้งสองโดยเขียนโปรแกรมทดสอบบนเครื่องคอมพิวเตอร์ยี่ห้อ ENCORE ซึ่งมี 6 โปรเซสเซอร์ปรากฏว่า อัลกอริทึมของผู้เขียนมีประสิทธิภาพสูงกว่าของบอคคาริมาก และเพื่อเป็นการเพิ่มประสิทธิภาพของอัลกอริทึมที่ผู้เขียนคิดขึ้น ผู้เขียนได้คิดวิธีหลีกเลี่ยงปัญหาที่ค่าโสหุ้ยในการปล่อยโปรเซสในแนวกั้นมีราคาสูง วิธีใหม่นี้ได้รับการพิสูจน์ว่ามีประสิทธิภาพสูงกว่าสองวิธีแรกมาก แต่เพื่อไม่ให้ยุ่งยากสับสนสำหรับผู้อ่าน จะขอไม่กล่าวถึงรายละเอียดในที่นี้ สำหรับผลการทดลองของอัลกอริทึมทั้งสามได้แสดงไว้ในรูปที่ 19 เวลาการคำนวณเมื่อ P = 1 นั้น วัดจากการ

จำนวนโปรเซสเซอร์	เวลาในการคำนวณ (วินาที)			
	ซีฟแบบอนุกรม	ซีฟแบบขนาน		
		วิธีของบอคคาริ	วิธีของผู้เขียน (1)	(2)
1	41.6			
2		23.8	20.2	19.2
3		17.8	14.2	13.2
4		15.0	11.4	10.8
5		14.0	9.8	9.3
6		13.6	8.8	8.2

รูปที่ 19 เปรียบเทียบเวลาในการคำนวณหาตัวเลขพรามทุกตัวในช่วง 2 ถึง 2 ล้าน ระหว่างวิธีของบอคคาริ และวิธีของผู้เขียนทั้งสองแบบ

คำนวณของอัลกอริทึมพื้นฐานที่ใช้สำหรับเครื่องแบบโปรเซสเซอร์เดี่ยว

จากตัวอย่างในตอนที่ 2 และตอนที่ 3 เราจะเห็นได้ว่าการเขียนโปรแกรมเพื่อการคำนวณแบบขนานนั้นยุ่งยากกว่าการเขียนโปรแกรมแบบอนุกรมมาก เพราะโปรแกรมเมอร์จำเป็นที่จะต้องรู้รายละเอียดของเครื่องคอมพิวเตอร์ที่ใช้อย่างลึกซึ้ง เนื่องจากอัลกอริทึมที่ออกแบบมาอาจจะทำงานได้ไม่ดีในเครื่องรุ่นนั้น ๆ นอกจากนี้การออกแบบอัลกอริทึมก็ยังยุ่งยากซับซ้อนมาก ดูจากตัวอย่างง่าย ๆ ที่ยกมาทั้งสองตัวอย่าง เรายังออกแบบอัลกอริทึมที่มีประสิทธิภาพสูงได้ยาก ทั้งยังต้องอาศัยเวลาและมีข้อปลีกย่อยให้พิจารณาอีกมากมาย ดังนั้น การออกแบบโปรแกรมเพื่อการคำนวณแบบขนานสำหรับปัญหาที่ยุ่งยากกว่านี้จึงทำได้ลำบาก บางวิธีที่เสนอขึ้นมา เมื่อดูตามทฤษฎีแล้วก็น่าจะทำงานได้ดี แต่เมื่อนำมาใช้งานจริง ๆ กลับมีปัญหาเรื่องค่าโสหุ้ยในการควบคุมต่าง ๆ ทำให้วิธีเหล่านี้มีประสิทธิภาพด้อยลง จนบางครั้งเลวกว่าประสิทธิภาพดั้งเดิมของโปรแกรมที่คำนวณบนเครื่องแบบโปรเซสเซอร์เดี่ยวเสียอีก

จุดประสงค์ของบทความชุดนี้คือจะแนะนำการเขียนโปรแกรมเพื่อการคำนวณแบบขนานสำหรับคอมพิวเตอร์แบบหลายโปรเซสเซอร์ แต่เครื่องคอมพิวเตอร์ดังกล่าวยังไม่แพร่หลายนัก ส่วนใหญ่ยังคงอยู่ในช่วงพัฒนาปรับปรุงหาข้อดีข้อเสียอยู่ อย่างไรก็ตาม มีแนวโน้มว่าในอนาคตเครื่องคอมพิวเตอร์แบบหลายโปรเซสเซอร์จะเป็นที่นิยมแพร่หลาย ไม่แพ้เครื่องแบบโปรเซสเซอร์เดี่ยว สำหรับประเทศไทย เครื่องคอมพิวเตอร์แบบนี้อาจจะเริ่มมีใช้ในเวลาไม่กี่ปีข้างหน้านี้ก็ได้

ถ้าผู้อ่านคิดว่าบทความประเภทนี้มีประโยชน์และสนใจอยากรู้เรื่องทางวิชาการคอมพิวเตอร์ในหัวข้ออื่น ผู้เขียนยินดีที่จะค้นคว้ามาเสนอแก่ผู้อ่านต่อไป นอกจากนี้ ถ้าผู้อ่านมีข้อเสนอแนะใด ๆ กรุณาติดต่อมาได้ที่

JINT WARAMAHAPUTI
USL BOX 43513
LAFAYETTE, LA 70504
U.S.A.

เอกสารอ้างอิง

1. บทความเรื่อง "Programing for Parallelism" โดย A.H. Karp จากหนังสือ Computer Magazine ประจำเดือนพฤษภาคม พ.ศ. 2530
2. บทความเรื่อง "Multiprocessing the Sieve of Eratosthenes" โดย S.H. Bokhari จากหนังสือ Computer Magazine ประจำเดือนเมษายน พ.ศ. 2530
3. บทความเรื่อง "Parallelization and Performance Analysis of the Cooley-Tukey FFT Algorithm for Shared-Memory Architecture" โดย A. Norton และ A.J. Silberger จากหนังสือ IEEE transactions on Computers ประจำเดือนพฤษภาคม พ.ศ. 2530
4. บทความเรื่อง "The Performance of Multicomputer Interconnection Network" โดย D.A. Reed และ D.C. Grunwald จากหนังสือ Computer Magazine ประจำเดือนมิถุนายน พ.ศ. 2530
5. หนังสือ "An Introduction to Operating Systems" โดย H.M. Deitel สำนักพิมพ์ Addison-Wesley Publishing Company พ.ศ. 2527
6. หนังสือ "List of Prime Number from 1 to 10,006,721" โดย D.N. Lehmer สำนักพิมพ์ Hafner Publishing Company พ.ศ. 2499
7. หนังสือ "Lecture Notes on Primality Testing and Factoring" โดย C. Pomerance สำนักพิมพ์ The Mathematical Association of America พ.ศ. 2527
8. บทความเรื่อง "Performance Evaluation of Parallel Prime Algorithms on Multiprocessing Systems," โดย จินต์ วรมหาภูติ และ S.P. Landry ส่งตีพิมพ์วารสาร IEEE Computer Magazine, พ.ศ. 2530